HIERARCHICAL CLUSTERING FOR AUTOMATIC FILE CLASSIFICATION: A STUDY ON SCALABLE APPROACHES

PROJECT REPORT

Submitted by

KAMESWARAN R - 2020239007

VISWARANJANI S - 2020239023

VIGNESHWARANN VIJAYAKUMAR - 2020242023

Submitted for

XC5951 – Advanced Machine Learning Techniques

DEPARTMENT OF MATHEMATICS

ANNA UNIVERSITY

CHENNAI 600 025

NOVEMBER 2024

ANNA UNIVERSITY

CHENNAI 600 025

BONA FIDE CERTIFICATE

Certified that this Project report titled **"HIERARCHICAL CLUSTERING FOR UNLABELED FILE CLASSIFICATION"** is the bona fide of work done by **Mr. KAMESWARAN R, Ms. VISWARANJANI S, Mr. VIGNESHWARANN VIJAYAKUMAR** in the **XC5951 – ADVANCED MACHINE LEARNING TECHNIQUES** Course during the Semester IX of the academic year 2024-2025.

Course Instructor

Table Of Contents

Hierarchical Clustering for Automatic File Classification:

A Study on Scalable Approaches

Student Names: Kameshwaran R, Viswaranjani S, Vigneshwarann Vijayakumar
Institution/Department: College Of Engineering Guindy,CEG/Mathematics.
Date:November 26, 2024

1. Objective

This study investigates a hierarchical clustering approach to classify unlabeled files based on their content and structure. It uses a Python program for feature extraction and clustering, focusing on computational efficiency and adaptability across various file types, including text, image, audio, and binary formats.

2. Paper Summaries

Title: A Rapid Review of Clustering Algorithms
Highlights the utility of hierarchical methods for datasets without predefined clusters and their sensitivity to input similarity metrics AR5IV.

Title: Hierarchical Clustering Objective Functions and Algorithms
Focuses on optimization criteria for hierarchical clustering and improvements over traditional methods like average-linkage AR5IV.

Title: A Study of Hierarchical Clustering Algorithms
Discusses the efficiency and scalability of algorithms like BIRCH and linkage-based approaches IEEE XPLORE.

3. Architectures and Methods:

i.Text Feature extraction (Hash-based approach):
Hash-based representation refers to the use of cryptographic hash functions (like MD5, SHA-256, etc.) to convert a block of text into a fixed-size string or vector of characters/numbers. This representation is deterministic, meaning

the same input will always produce the same hash, and it is compact, regardless of the original text size. This representation is lightweight and computationally efficient, making it suitable for high-level comparisons, duplicate detection, or identifying similar files without delving into detailed semantic or structural features.

ii.Image feature extraction:

This process refers to the transformation of image data into a standardized format by converting it to grayscale and resizing it to a fixed dimension. It is a common preprocessing step in image analysis and machine learning workflows to ensure consistency and reduce computational complexity.

iii.Unknown/Other Files Feature extraction:

Combining file size and hash-based encoding is a technique used to create a unique numerical or vector-based representation of a file. The combination of these two features helps capture both structural and content-specific information about the file, which can be useful for various tasks such as file classification, similarity analysis, and file identification.

iv.Audio Feature extraction:**(Spectrogram)**

Spectrogram feature extraction refers to the process of converting an audio signal into a visual representation of its frequency content over time (a spectrogram) and then using that spectrogram to derive meaningful numerical features that can be used for further analysis, such as classification, clustering, or machine learning tasks.

Mathematical models

Jaccard Similarity:

The Jaccard Similarity (or Jaccard Index) is a statistical measure used to quantify the similarity and diversity between two sets.In this project Jaccard Similarity is used to compare files based on their feature representations. Since we are dealing with different file types (text, images, audio, etc.), these

features might represent different types of information, such as words in text files, pixel intensities in images, or byte frequencies in binary files.

Interpretation:

Jaccard similarity ranges from 0 to 1

- 0 means the sets have no elements in common (completely dissimilar).
- 1 means the sets are identical (completely similar).
- Values between 0 and 1 indicate varying degrees of similarity.

Evolution technique:

In traditional file classification methods, files are often manually labeled or categorized based on predefined rules or features. These approaches can involve

Supervised Learning: Training machine learning models (e.g., decision trees, SVMs) on labeled datasets to predict the category of a file. Each file is assigned to a category based on features extracted from the file's content (e.g., text, metadata, or visual features).

Rule-Based Systems: Classifying files based on predefined rules or conditions. For example, files might be categorized by extension (e.g., .txt files as documents, .jpg files as images).

Manual Categorization: Human experts manually assign files to categories based on content or metadata, which can be time-consuming and impractical for large datasets.

hierarchical clustering: Hierarchical clustering is an unsupervised clustering method that automatically classifies unlabeled files into categories based on their content similarity, without the need for predefined labels.

4.Dataset Used:

The dataset used for this analysis comprises a diverse collection of randomly selected files of varying types and sizes. These include text files, images, audio files, and binary files, each contributing unique characteristics to the clustering process. Text files are represented using hash-based vectors, while images are converted to grayscale and resized to ensure uniformity. Audio files are processed into spectrogram features, and binary files are analyzed through their byte frequency distributions. This heterogeneous dataset serves as a robust testbed for evaluating the effectiveness of hierarchical clustering algorithms in categorizing files based on their intrinsic properties.

1.Image Dataset:

Image File Dataset (JPG, PNG)

Sources: A collection of publicly available images in various formats (JPG, PNG) and some synthetically generated samples.

Size: Approximately 200 images in different formats, including both high quality and low-quality images.

Pre-processing Steps: Images were resized to 64x64 pixels, converted to grayscale for consistency, and then flattened into one-dimensional vectors for feature extraction.

2.Document Dataset:

Document File Dataset (Word, PowerPoint, PDF)

Sources: A collection of documents in various formats such as Word (.docx), PowerPoint (.pptx), and PDF files, sourced from publicly available document repositories and a synthetic dataset.

Size: Approximately 150 document files of varying content (e.g., reports, presentations, manuals).

Pre-processing Steps: Documents were parsed and converted to text using file extraction libraries. The text was then transformed using TF-IDF vectorization or hash-based feature extraction.

3. Programming Files Dataset:

Programming Language Files Dataset (Python, Jupyter Notebooks, C++)

Sources: A collection of source code files in Python (.py), Jupyter Notebooks (.ipynb), and C++ (.cpp) formats, gathered from public code repositories and synthetic code samples.

Size: Approximately 100 programming files in different languages.

Pre-processing Steps: Code files were parsed into textual representations, with tokenization and feature extraction done using hash-based or TF-IDF techniques to capture programming-specific features.

4.Combined Multi-format Dataset:

Multi-format Dataset (Images, Documents, Programming Files)

Sources: A mix of image files, document files, and programming code files from the above datasets.

Size: A total of 500 files, evenly distributed across the three categories.

Pre-processing Steps: Each file was preprocessed according to its type (e.g., text extraction for documents, image resizing for image files, and tokenization for programming code). Each file type was represented as a vector for clustering

5. Evaluation Metrics:

- **Silhouette Score:** The Silhouette Score is a metric used to evaluate the quality of clustering in data. It measures how similar a data point is to its own cluster compared to other clusters. The score helps determine the appropriateness of the clustering results, where a higher score indicates well-defined clusters.

$$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

- **Davies-Bouldin Index (DBI):** The Davies-Bouldin Score is a metric used to evaluate the quality of clustering results by analyzing how well-separated and compact the clusters are. A lower Davies-Bouldin Score

indicates better clustering, as it reflects clusters that are compact and well-separated from one another.

$$DB = \frac{1}{n}\sum_{i=1}^{n}\max_{i\neq j}\left(\frac{\sigma_i + \sigma_j}{d_{ij}}\right)$$

- **Intra-cluster Distance:** Intra-cluster distance measures how close the data points within a single cluster are to each other. It is typically calculated as the distance between each data point and the cluster centroid. The smaller the intra-cluster distance, the more similar the data points are within that cluster.

$$\text{Intra-cluster Distance} = \frac{1}{n}\sum_{i=1}^{n}\|x_i - \mu\|$$

- **Inter-cluster Distance:** Inter-cluster distance refers to the measure of separation between different clusters in a clustering analysis. This metric is crucial for evaluating the quality and effectiveness of clustering algorithms, as it helps determine how distinct and well-separated the clusters are from each other.

$$\text{Inter-cluster Distance} = \frac{1}{k(k-1)}\sum_{i=1}^{k}\sum_{j=i+1}^{k}\|\mu_i - \mu_j\|$$

- **Dendrogram Analysis:** Provides a visual method for evaluating and interpreting the clustering hierarchy.

6. Code Replication and Result Verification:

6.1 Environment Setup:

- **Tools**: Python (Scikit-learn, SciPy, Matplotlib), TensorFlow for spectrogram processing.
- **Hardware**: Standard CPU environment.

6.2 Execution Steps:

Data Preparation:

Gather all files to be clustered into a single directory. Process each file to extract meaningful features based on its type

- **Text Files:** Represent as numerical features using hashing or text vectorization methods.
- **Image Files:** Convert to grayscale, resize to a standard size (e.g., 64x64), and flatten into a vector.
- **Audio Files:** Extract basic audio features, like spectrogram data.
- **Other Files:** Use byte frequency or file hashes to represent their content.

Feature Engineering:

- **Dimensional Adjustment:** Pad or truncate the extracted feature vectors to ensure uniform length across all files.
- **Normalization:** Normalize features (if applicable) to ensure consistency in magnitude for clustering algorithms.

Similarity Computation:

- **Similarity Metric:** Compute pairwise Jaccard similarities between feature vectors.
- **Distance Matrix:** Convert similarity scores to a distance matrix using 1 − similarity 1−similarity for compatibility with hierarchical clustering.

Hierarchical Clustering:

- **Algorithm:** Use agglomerative clustering with a selected linkage criterion (e.g., average linkage).
- **Linkage Matrix:** Compute the linkage matrix as an intermediate representation of the clustering hierarchy.

Visualization of dendrogram:

Truncate filenames for readability. Visualize the clustering process and cluster separations using a dendrogram. Use appropriate labels and thresholds to improve interpretability.

Cluster Analysis:

- **Intra-Cluster Metrics:** Compute centroids for each cluster. Measure intra-cluster distances (average distance of points to the cluster centroid).
- **Inter-Cluster Metrics:** Calculate distances between cluster centroids. Derive the average inter-cluster distance for assessing cluster separability.

Validation and Evaluation:

- **Silhouette Score:** Evaluate clustering quality by measuring how similar objects in the same cluster are compared to objects in other clusters.
- **Davies-Bouldin Index:** Calculate a score to assess intra-cluster compactness and inter-cluster separation.

6.3 Comparison of results:

The results demonstrated that Jaccard similarity was the most effective measure for hierarchical clustering in file classification, outperforming cosine similarity and Euclidean distance. Jaccard's focus on the ratio of shared features to total unique features proved ideal for sparse and categorical file data, producing more intuitive and meaningful clusters. Cosine similarity, while effective in vector spaces, struggled with sparse data, and Euclidean distance overemphasized differences, leading to less cohesive groupings. Additionally, average linkage was employed for cluster merging due to its ability to balance intra-cluster similarity and inter-cluster separability. This made it more suitable than single linkage, which can form elongated clusters, or complete linkage, which can be overly restrictive. Together, Jaccard similarity and average linkage ensured well-defined clusters aligned with the structure of the file dataset.

7. Results and Discussion:

Results:

- **Effective Clustering for Homogeneous File Types:** The program successfully grouped files with clear and distinct features. For example: Text files with similar content or structure were clustered accurately. High-quality images with similar visual patterns or objects were grouped together. This demonstrates the capability of the feature extraction methods in capturing meaningful representations.
- **Challenges with Mixed or Ambiguous Data:** Mixed clusters were observed for files with less-defined or noisy representations. For example: Low-quality or blurry images led to less accurate feature extraction, resulting in misclassification. Unsupported or poorly structured files, such as partially corrupted binary files, showed ambiguous clustering behavior.
- **Dendrogram Insights:** The generated dendrogram effectively illustrated hierarchical relationships between clusters, providing a clear visualization of file similarities. Users could identify logical groupings, as well as outliers or files that were not strongly associated with any cluster.

Discussion:

Strengths of the Custom Program:

- **Adaptability to File Types:** The program handled a wide variety of file formats, showcasing flexibility over traditional methods like BIRCH, which are typically optimized for structured data. Tailored feature extraction ensured better representation for each file type.
- **Visualization with Dendrograms:** The dendrogram provided an intuitive overview of the clustering process, aiding in understanding and analysis.

Manual Parameter Tuning:

- **Feature Extraction Parameters:** Optimal results required manual adjustments, such as resizing dimensions for images, selecting hash functions for text, or tuning similarity thresholds. This dependency limits scalability and user-friendliness for non-technical users.

- **Thresholds for Clustering:** The static thresholds used for dendrogram pruning were not universally effective, leading to less precise cluster separations in some cases.

8. Pros and Cons:

Pros:

- **Customizability**: The program is designed to handle multiple file types, including text, images, audio, and binary files. Feature extraction techniques can be adapted or fine-tuned based on file type, ensuring more accurate representations and clustering.
- **Scalability**: Handles moderate-sized datasets efficiently by leveraging modular processing and optimized similarity computations. Clustering algorithms such as hierarchical clustering can work well even when new file types are added.
- **Visualization**: Dendrograms provide an intuitive representation of the clustering hierarchy, making it easier to identify groupings and patterns. Truncated labels and customization options improve the readability of results, especially for large datasets.

Cons:

- **Complexity**: The system requires significant domain-specific knowledge to design preprocessing pipelines for various file types. Users must understand the nuances of feature extraction, such as converting images to grayscale or handling text encoding.
- **Performance**: The hierarchical clustering algorithm, while robust, has a higher computational cost compared to simpler approaches like K-means. For large datasets, the similarity computation and dendrogram plotting can become resource-intensive.
- **Sensitivity**: The quality of clustering heavily depends on the choice of similarity metrics and linkage methods. Slight variations in parameter settings (e.g., resizing dimensions for images or threshold values for clustering) can significantly impact the results, making the system sensitive to input adjustments**.**

9. Conclusion and Future Work:

Conclusion:

The custom hierarchical clustering program demonstrates notable flexibility and accuracy in classifying diverse file types, including text, image, audio, and binary data. By leveraging tailored feature extraction methods and hierarchical clustering, it effectively groups similar files and provides intuitive visualizations through dendrograms. However, the system's performance is dependent on manual adjustments and parameter tuning, indicating room for optimization. Overall, this project showcases the potential of hierarchical clustering for managing heterogeneous file datasets and provides a solid foundation for future improvements.

Future Work:

Integrating Deep Learning Methods for Feature Extraction:

- **Current Limitation:** The program relies on handcrafted feature extraction techniques, which may not capture complex patterns, especially in ambiguous or noisy data.
- **Future Direction:** Incorporating deep learning models, such as convolutional neural networks (CNNs) for images or transformers for text, can enable automatic and more robust feature extraction.

Enhancing Scalability with Distributed Processing:

- **Current Limitation:** The clustering process, particularly the computation of pairwise similarities and hierarchical linkage, can become computationally expensive for large datasets.
- **Future Direction:** Implement distributed computing frameworks like Apache Spark or TensorFlow Distributed to parallelize feature extraction and similarity computation.
- **Impact:** This would allow the system to efficiently handle larger datasets and scale to enterprise-level file classification tasks.

Exploring Dynamic Thresholds for Dendrogram Pruning:

- **Current Limitation:** The selection of clustering thresholds (e.g., distance or similarity thresholds) is static and requires manual experimentation.

- **Future Direction:** Develop dynamic thresholding methods that adapt based on the dataset's characteristics, such as the density or distribution of file features.
- **Impact:** Dynamic thresholds would enable more accurate cluster separations, especially for datasets with varying levels of similarity, and reduce the need for manual intervention. By addressing these areas, the program could evolve into a more autonomous, scalable, and robust solution for file classification in diverse domains.

10. References:

1. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer. https://doi.org/10.1007/978-0-387-84858-7

2. Aggarwal, C. C., & Reddy, C. K. (2013). Data Clustering: Algorithms and Applications. CRC Press. https://doi.org/10.1201/b15405

3. Scikit-learn. (n.d.). Scikit-learn: Machine Learning in Python. Retrieved from https://scikit-learn.org

4. Virtanen, P., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods, 17(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

5. van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering, 13(2), 22–30. https://doi.org/10.1109/MCSE.2011.37

6. BIRCH Algorithm. (1996). Zhang, T., Ramakrishnan, R., & Livny, M. BIRCH: An Efficient Data Clustering Method for Very Large Databases. ACM SIGMOD Record, 25(2), 103-114. https://doi.org/10.1145/233269.233324

7. OpenAI. (n.d.). OpenAI: Machine Learning Research and Applications. Retrieved from https://openai.com